Dkt. 0655/64696

Application

for

United States Letters Patent

To all whom it may concern:

Be it known that,

Richard H. HARVEY

have invented certain new and useful improvements in

DIRECTORY SEARCHING METHODS AND SYSTEMS

of which the following is a full, clear and exact description:

DIRECTORY SEARCHING METHODS AND SYSTEMS

CROSS-REFERENCE TO RELATED APPLICATIONS

The present application is a continuation-in-part of U.S. Serial No. 09/427,267 filed October 26, 1999, which is a divisional of U.S. Serial No. 08/793,575 filed February 28, 1997 (now US Patent 6,052,681), which is a National Stage of International Application No. PCT/AU95/00560 filed August 30, 1995, each of which are incorporated herein in their entirety by reference.

BACKGROUND

1. Field

The present application relates to the field of directory services. More particularly, the present application relates to the application of electronic directory services, e.g., X.500 or LDAP, in relational databases, to table structures in database arrangements used for searching, and to methods for searching databases.

2. Description of the Related Art

A Relational Database Management System (RDBMS) provides facilities for applications to store and manipulate data. Amongst the many features that an RBDMS offers are data integrity, consistency, concurrency, indexing mechanisms, query optimisation, recovery, roll-back, security. RBDMS also provide many tools for performance tuning, import/export, backup, auditing and application development.

RDBMS are the preferred choice of most large scale managers of data. RDBMS are readily available and known to be reliable and contain many useful management tools. There is a large base of RDBMS installations and therefore a

large amount of existing expertise and investment in people and procedures to run these systems, and so data managers are looking to use this when acquiring new systems. Most relational database products support the industry standard SQL (Structured Query Language).

There has also been a move towards Object Oriented systems, which provide data extensibility and the ability to handle arbitrarily complex data items. In addition, many corporations and governmental departments have a large number of database applications, which are not interconnected. Data managers are looking for solutions which enable them to integrate their data, and to simplify the management of that data. Electronic directories provide data managers with a tool to achieve these objectives. Some electronic directories are standardized. X.500 is the International Standard for Electronic Directories [CCITT89 or ITU93]. These standards define the services, protocols and information model of a very flexible and general purpose directory. X.500 is applicable to information systems where the data is fairly static (e.g. telephone directory) but may need to be distributed (e.g. across organisations or countries), extensible (e.g. store names, addresses, job titles, devices etc.), object oriented (i.e. to enforce rules on the data) and/or accessed remotely. Electronic directories, such as X.500 and its associated standards, provide a framework and a degree of functionality that enables data managers to achieve their objectives.

Typically, data managers prefer to implement an electronic directory, e.g., an X.500 directory, with all the flexibility of object-oriented systems, but using an SQL product so that the system can achieve the scalability and performance

inherent in relational systems coupled with the stability, robustness, portability and cost-effectiveness of current SQL products.

One example of an electronic directory implementation is described in U.S. Serial No. 09/427,267 and its corresponding Australian Patent 712451 both of which are incorporated herein in their entirety by reference. In the search strategies for this implementation, at a conceptual level, hierarchy tables, e.g., NAME, DIT and TREE, are used to maintain relationships between objects in a hierarchy. These hierarchy tables are arranged according to one row per object. Object tables, e.g., SEARCH and ENTRY, manage values within an object. These object tables are arranged according to one row per value. Every object has a corresponding row in the hierarchy tables and every attribute value has a corresponding row in the object tables. In this implementation, the object table used for searching objects contains rows of the form (EID, AID, VID, Norm), where EID identifies the object to which the value belongs, AID identifies the attribute type of the value, VID identifies one of a possible number of attribute values in the one entry, and Norm contains the syntax normalized value. Attribute tables, e.g., attribute, define information about attribute types. The attribute table used contains rows of the form (AID, SYX,DESC, OBJECT ID).

It has been discovered that improvements to electronic directory implementations may be achieved for arranging and searching databases for complex data types.

SUMMARY

The present application relates to the storing and/or searching for data types using some form of indicia, such as one of the components contained in stored data, an identifier of stored data and / or a manipulation of stored data. This implementation facilitates the searching of complex data types by adding new search and/or attribute tables that store information relating to data entries, predetermined information considered to be of assistance or useful in searching for particular entries of a database, such as, by not limited to including component identifier (CID) information representing an individual component and/or component value identifier information (CVID) representing a multi-valued component or components used for searching. These new tables are referred to herein as subsearch tables and/or subattribute tables which serve to store components of values, and facilitate the searching of individual components and/or multi-valued components. However, such tables can be referenced with any name.

In one embodiment, the present application provides a method of arranging data in a database. The method includes creating a first table adapted for storing the data and having one row for each data entry, and creating a second table adapted for storing data components and having one row for each component of the stored data type. Preferably, the data is a structured data type or a string data.

The present application also provides a database and / or directory having a data storage arrangement. Throughout the remainder of the specification, reference is made to a database, however this equally applies to a directory

services system. The data storage arrangement includes a first table directed to a hierarchy which defines a relationship between objects and configured to have one row per object, a second table directed to objects which define one or more values within each object and configured to have one row per value, and a third table directed to one or more selected components or representations of values and configured to have one row for each component of each value. Preferably, the database is a part of a directory services system, such as X.500 or LDAP services system.

The present application also provides a method of searching a database for a given data entry. In this embodiment, the database has a first table adapted for storing data and having one row for each data entry, and a second table adapted for storing data components or representations and having one row for each component of the stored data. The searching method includes determining a component or representation of a given data entry, executing one of an exact or initial matching on the second table in order to locate the component or representation, and returning the given data entry matching the component or representation located.

Throughout the specification, reference to 'component' may instead or in addition include 'representations' of values, e.g. reverse indexing or pointers or fingerprints or checksum, or some suitable smaller representation of relatively large data.

BRIEF DESCRIPTION OF THE DRAWINGS

Preferred embodiments of the present application will now be described with reference to the accompanying drawings, in which

Figure 1a illustrates a principle and conceptual design of an exemplary database according to the present application;

Figure 1b illustrates a logical and physical design of an exemplary database according to the present application including a subsearch table and a subattribute table;

Figure 2 is a schematic representation of a high order search method according to the present application;

Figure 3 is a schematic representation of a sub-order search method according to the present application;

Figure 4 illustrates an exemplary X.509 certificate and corresponding example of search and subsearch tables; and

Figure 5 illustrates exemplary search and subsearch tables related to a telephone number data component.

DETAILED DESCRIPTION

Figure 1a illustrates an exemplary implementation of a database design and is only one type of database to which the methods and arrangements of the present application can be applied. A more detailed description of this database design can be found in U.S. Serial No. 09/427,267 which is incorporated herein in its entirety by reference. Figure 1b illustrates a more detailed implementation of the database design of Figure 1a. Generally, when searching a directory having a database, a search argument defines where to start the search (base object), the scope of the search (subset), the conditions to apply to the search (filter) and what information should be returned from the search (selection). The filter is a

combination of one or more filter items (or conditions) connected by operators such as AND, OR and NOT.

For general search services filters are applied to values and attributes in the object table, e.g., the search table. An example of a general filter is "NORM LIKE '%RICK HARVEY%'". For particular search services (described in more detail below) a clause may be added the SQL statement that includes a component identifier, which addresses a particular component of a data type and an exact or initial (or "begins with") filter is applied to components in a subsearch table instead of the search table. An example of such a clause is "AND CID = n", and an example of an exact filter is "NORM = 'RICK HARVEY'".

Referring to Figure 1b, the table structure of logical design 1 and physical design 2 include search table 3, subsearch table 4, attribute table 5, and subattribute table 6. The search table 3 and attribute table 5 are similar to the search and attribute tables described in U.S. Serial No. 09/427,267 and will not be described in more detail here. The subsearch table 4 can be configured to include one or more components used to improve the speed and reliability of the search. For example, the subsearch table 4 can include a CID field 7 and a CVID field 8. The CID field 4 serves as a tool (or index) for searching components of data types, and the CVID field 8 serves as a tool (or index) for multi-valued component searching. Although the method according to the present application can be used with one or more of the above-identified tables, preferably, each table is provided so that there is a choice for particular search queries.

The attribute table 5 describes or references information in search table 3, and the subattribute table 6 describes or references information in subsearch

table 4. The subattribute table 6 has similar fields as the attribute table 5, but substitutes CID field 9 for AID field 10. The CID field 9 is used to identify one or more components in the subsearch table 4.

The subsearch table 4 preferably stores information that improves searching performance or components of complex data types. Other components stored in the subsearch table can be those that improve the manageability of the database. In other words, it is not a requirement that every value in a data entry is included in the subsearch table.

Referring to Figures 2 and 3, methods for searching a database will be described. Examples of searches include base object and whole tree searches 11, one level search 12 and subtree search 13. More detail descriptions of these searches can be found in U.S. Serial No. 09/427,267. Searches that have a scope of base object or whole tree 11 use the search table 3. Searches that have a scope of base object or whole tree 11 and can make use of a stored component use the subsearch table 4. Searches that have a scope of one level 12 use a join between the DIT table 14 and search table 3. Searches that have a scope of one level 12 and can make use of a stored component use a join between the DIT table 14 and subsearch table 4. Searches that have a scope of subtree 13 use a join between the tree table 15 and search table 3. Searches that have a scope of subtree 13 and make use of a stored component use a join between the tree table 15 and subsearch table 4.

To illustrate, if a desired search argument is structured with a general filter, the base object and whole tree searches 11 would use the search table 3, the one level search 12 would use DIT table 14 and search table 3, and the subtree

search 13 would use tree table 15 and search table 3. An example of an SQL statement for such a search may be:

SELECT EID{other columns} FROM SEARCH {other tables} WHERE {filter}

However, in such a search, if the filter applied were, for example, to a highly repetitive data string, a portion of a data string or a component of a structured data type, the search may be slow and inefficient because the database may have to scan through a large number of values.

One way to improve the efficiency of such a search is to utilize a subsearch table and search one or more components associated with the stored data. In such instances the database may be able to use an index to the component in the string or structure thus avoiding a scan of a large number of values. An example of an SQL statement for such a search would be:

SELECT EID {other columns} FROM SUBSEARCH {other tables} WHERE {filter} AND CID=x

In this implementation, the base object and whole tree searches 11 would use the subsearch table 4, the one level search 12 would use DIT table 14 and subsearch table 4, and the subtree search 13 would use tree table 15 and subsearch table 43. In this example, the filter used would reference a component of the data stored, which permits the use of an index and results in faster more efficient searches. The index used in this example includes CID=x.

Structured Attributes

Methods according to the present application can be used in various other applications. One application is the security area where directories are increasingly being used by Certification Authorities to store standardized certificates. An example of such a certificate is an X.509 certificate. Certificates such as the X.509 can be referred to as 'complex' attributes because they contain many components. However, the present application should not be limited to only these types of certificates. It will be understood that the present application can be utilized with any form of information having components.

When storing such certificates consideration should be given as to how the certificate is stored so that retrieval of a certificate is quick and reliable (i.e., the desired certificate is actually retrieved). The methods and database arrangements of the present application achieve this by finding and managing one or more of the components of the data in the certificate, e.g., serial number, expiration date and issuer.

Figure 4 illustrates the application of the present application to X.509 certificates. For the purposes of clarity only a small part of each table is illustrated. The certificate 20 is illustrated schematically, and for the purposes of illustration only shows information, such as the issuer at field 21, validity information at field 22, serial number at field 23, version number at field 24, and subject information (e.g., rick harvey) at field 25. For this example, the search table 3 is arranged in one row with spaces (preferably two) separating the normalized value of each component or field of the certificate 20. The search table also includes a normalized value representing the entire certificate. The

subsearch table 4 is arranged with one or more rows (26, 27, 28, 29 and 30) where one row corresponds to one component or field in certificate 20. It should be noted however that the subsearch table 4 does not have to include every field identified in certificate 20.

To illustrate an implementation for this embodiment, assume that a simple certificate consists of information similar to what a credit card holds, e.g., a serial number, an expiration date, and the cardholders name. This simple certificate has three components or fields, namely a number field, a date field and a string field. In this simplified example, the normalized value of certificate 20 that would be stored in the search table 3 (of the form in Figure 1b) is as follows:

(xx, yy, zz, "123456 20000806123000 RICK HARVEY)"

and the subsearch table 4 may store, for example, three rows - one for each component of the certificate. Each row of the subsearch table would be in the form of Figure 1b:

(xx, yy, zz, 0, 0, "123456")
(xx, yy, zz, 1, 0, "20000806123000")
(xx, yy, zz, 2, 0, "RICK HARVEY")

where xx, yy and zz are integers corresponding to fields in the particular table design, such as EID, AID and VID.

A search for a certificate that was issued to "RICK HARVEY" that utilized the search table 3 may use the following SQL statement:

SELECT ... FROM ... SEARCH ... WHERE AID = 27 and NORM LIKE '%RICK HARVEY%'

where the general filter is "NORM LIKE '%RICK HARVEY%'". However, using search table 3 for such a search may be slow because each component of the string for each entry would have to be examined and the degree of certainty that the string being searched and retrieved is the desired entry. This is because the flattened representation has no boundaries as it is an unstructured text representation.

A search for a certificate that was issued to "RICK HARVEY" would be more efficient if subsearch table 4 were utilized when applying an exact or initial filter and a component identifier were added to the search argument. In this instance, the following SQL statement may be used:

SELECT ... FROM ... SUBSEARCH ... WHERE ....AID = 27 AND CID = 4 AND NORM = 'RICK HARVEY'

Where "NORM = 'RICK HARVEY'" is the filter, AID = 27 is the attribute identifier for a certificate and CID = 4 is the component identifier for the subject.

In the above example, a search with the component identifier (or index) CID=4 is used because it is known from the design of subsearch table 4 or from subattribute table 6, that index CID=4 is a string representing cardholders name. A query where the index CID is set to 4, and the filter is "NORM = 'RICK HARVEY'" should return Rick Harvey's certificate. This query is considered to be better because it can make use of an appropriate index making the search faster and increasing the degree of certainty that the search will find the correct certificate or certificates. It should be noted that the actual designation of characters/letter or numerals in the subsearch table design is arbitrary, and may be designed in whatever manner to suit the particular application.

Referring again to Figure 4, an alternative implementation of a method according to the present application will be described. In this implementation, search table 3A is a search table arranged to include a check sum or finger print. Because the search table 3A is used for exact matching on some attribute types e.g. binary, the value in the search table may be a fingerprint or checksum value. This makes the storage in the search table more efficient, as there is less data required to be stored.

String Attributes

The present application can also be utilized for non-complex data types, such as string data types. Examples of string data types include multi-word sentences, multi-line paragraphs of text, and a multi-line postal addresses. In this case, an attribute value that is a simple sentence may be stored in a single row in the search table as:

(1122, 33, 0, "MANY WORD SENTENCE")

where columns (or fields) are defined as (EID, AID, VID, NORM). A query searching the string data type for 'WORD' would involve looking at the rows for the part-word "%WORD%, which is considered a relatively slow search. To improve searching of such data the string is stored as components in the subsearch table 4 so that the string "MANY WORD SENTENCE" would be stored in three rows as follows:

(xx, yy, zz, 0, 0, "MANY")
(xx, yy, zz, 0, 1, "WORD")
(xx, yy, zz, 0, 2, "SENTENCE")

where the columns are defined as (EID, AID, VID, CID, CVID, NORM), respectively. In this example, the filter applied would then use the subsearch table 4 instead of the search table 3 and the following SQL statement could be used to search for “WORD”:

```
SELECT ... FROM SUBSEARCH ... WHERE AID = 33 AND CID = 0 AND
NORM = ”WORD”
```

The result would be a much faster search as we are looking for an exact match of “WORD” rather than, as above, looking for a part word “%WORD%”.

<u>Alternative Index</u>

The present application also has general applicability to the problem of being able to add one or more indices to a given attribute for the purpose of increasing performance for certain types of queries. Adding indices provides a different path in order to find an attribute, such as for example, reverse indexing. In this case, there may only be one component in the subsearch table. The component in effect represents an alternate form of that attribute value.

In particular, the subsearch table can cope with the problem of "ends in" searches or searching for values where an initial portion of data stored is relatively highly repetitive (such as distinguished names, MAC addresses, telephone numbers, full qualified file names, etc) by storing a reversed form of the value and thereby giving effect to having a reversed index on the attribute. For example, referring to Figure 5, a telephone number 31 is entered into a search table 32, and the telephone number is also entered into a subsearch table 33 in a reverse form. Of course this aspect of application is not limited to the reverse form, it may be any other suitable form of data entry suitable for a given

situation or search, such as treating the area code of a telephone number as a separate component.

When searching for a telephone extension, which is typically the end portion of a telephone number, a search may be expressed as a string search for "*1234" (the star being a wild card). If only search table 32 were used, then the performance of the search would be slow because indexes are only possible for "begins with" or "exact" searches. However, with the attribute stored in the subsearch table 33 in reverse, the subsearch table could be used to do an equivalent search, e.g. "4321*" which is considered to be very fast.

More particularly, the search table 32 might store a telephone number for a given person as:

(1122, 44, 0, “98791234”)

and in the subsearch table 33 would be stored:

(1122, 44, 0, 0, 0, “43219789”)

and an SQL statement to find a telephone number that ends in “1234” could be of the form:

```
SELECT … FROM SUBSEARCH … WHERE AID = 44 and CID=0 and
NORM LIKE ‘4321%’
```

Searching the subsearch table 33 should retrieve the desired record faster than searching the search table 32 because the search is for an initial match (i.e., begins with) for “4321”, which is a faster search.

Another example of an alternative index is the storing of a checksum of a binary value, e.g., photograph or audio.

Still another example of an alternative index is the storing of a fingerprint being a smaller representation of the data, such as a smaller representation of a photograph.

Although the present application is disclosed with reference to an X.500 directory services system, the application should not be limited to the system and methods disclosed therein. As will be understood from a reading of the specification as a whole, the present application may be applied or implemented in a number of different directory services systems or use a variety of methods.